

## /trends &amp; hypes

# Babylonische spraakverwarring

**De abacus is een telraam dat zeker al 2500 jaar terug in Egypte werd gebruikt. Mogelijk gaat het rekenhulpje zelfs terug tot 3000 voor Christus. Sporen leiden naar Babylonië. De bron van de oercomputer, maar ook de bron van spraakverwarring.**

Er komen met grote regelmaat nieuwe computertalen beschikbaar. De vraag voor elk bedrijf is wat daarmee te doen. Alles volgen is ondoenbaar, daarvoor ontbreken tijd en geld. En het is ook niet verstandig, omdat lang niet elke taal een lang leven is beschoren. Talen beoordelen en vergelijken vergt inzicht en kennis van de toepassing, de software en de vocabulaire. Sterke en zwakke typing is wat anders dan statische en dynamische typing. Maar wat is dat dan precies? En wat zegt het over een taal, over de productiviteit van de programmeurs, de flexibiliteit, kwaliteit en houdbaarheid van het eindproduct? Op die vragen probeert dit artikel een antwoord te geven. In de hoop de keuze voor de juiste taal te vergemakkelijken. Want vernieuwen is noodzakelijk om mee te kunnen doen in de eeuwige race naar steeds grotere en complexere systemen. Consolidatie/standaardisatie kan simpelweg niet.

## Static en dynamic typing

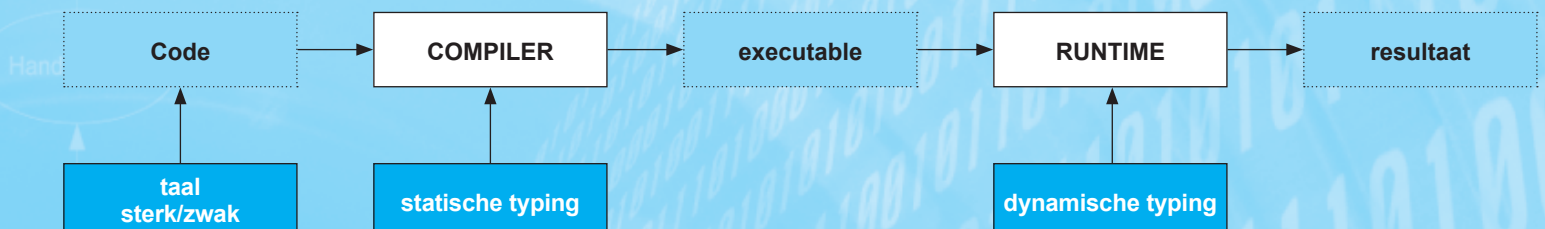
Een taal is een brug tussen de mens en de computer. Daarin vertelt de mens wat de computer moet doen. De basis voor elke taal is hetzelfde: de gebruiker schrijft een programma als een brok tekst (broncode). Dat gaat door de compiler, die de tekst vertaalt naar computertaal (de executable). Bij sommige talen controleert voornamelijk de compiler op fouten in de code. Dit heet statische typing. Bij andere talen controleert de compiler niet zo veel, maar komen we de fouten voornamelijk tegen bij het uitvoeren van de code. Dat is een runtime error. Een taal die de controle op fouten overwegend uitbesteedt aan de runtime, is een dynamisch getypeerde taal.

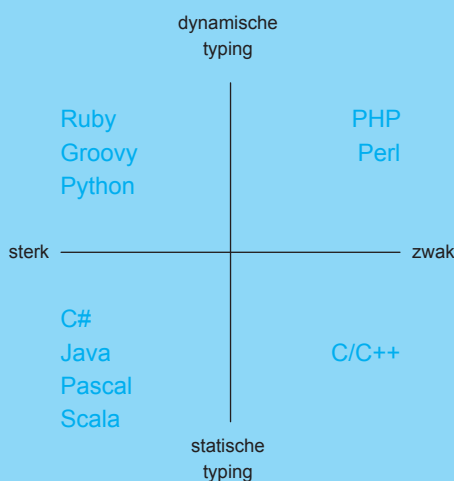
## Wat is een type?

Het begrip 'type' duidt op het soort data dat een programma verwerkt. Een programmeur kan niet zomaar elke bewerking op elk soort data loslaten. Hoe strikt een taal met types omgaat, wordt aangeduid met sterke dan wel zwakke typing. Statische en dynamische typing staan dus los van de begrippen sterke en zwakke typing. Dat veroorzaakt soms een babylonische spraakverwarring: je kunt sterke en zwakke dynamische typing hebben, en sterke en zwakke statische typing. Voordat we ook hier in een babylonische spraakverwarring belanden, maken we dit duidelijk met een analogie.

## Appels en peren

Laten we een computerprogramma voorstellen als een fruitkraam. In een zwakke taal mag al het fruit door elkaar liggen in de fruitkraam. In een sterke taal moeten alle appels in de appelkist en alle peren in de perenkist. Een kist is dan het equivalent van een array, die je in het zwakke geval met alles mag vullen en in het sterke geval bijvoorbeeld alleen met reële getallen. In een zwakke taal kunnen er problemen ontstaan wanneer al het fruit in zo'n gemengde fruitkist dezelfde bewerking moet ondergaan. Dat zal niet gebeuren met een sterke taal als Java. Die eist eerst definities van alle soorten fruit, om daarna de bewerking 'pellen' alleen aan bananen toe te kennen, en 'schillen' alleen aan appels en peren. In een zwakke taal (zoals C) kan de programmeur makkelijk allerlei types mixen, maar dat kan ook weer allerlei onbedoelde effecten hebben en uiteindelijk uitmonden in een programmacrash.





#### Taal Web framework

C#	ASP.NET
Java	JavaEE, Tomcat, Struts2, Tapestry, Wicket
Python	Django
Ruby	Rails
Groovy	Grails
Scala	Lift
Perl	Catalyst
PHP	Zend Framework

### Hello world

Een manier om een eerste indruk te krijgen van een taal met betrekking op de gebruiksvriendelijkheid is hoeveel code nodig is om de tekst "hello world" op het beeldscherm te zetten. In de ene taal is één regel genoeg, in andere talen vergt dat zowat een pagina met code. Het is een indicatie voor het gemak van instappen. Kan ik er zo mee aan de slag of moet ik eerst van alles leren? Dynamisch getypeerde talen hebben een lage instap-drempel en zijn veel flexibeler in gebruik. Statisch getypeerde talen bieden meer garantie op foutvrije software omdat de compiler de meeste fouten al opspoot. Maar ze zijn ook complex en bewerkelijk. De programmeur moet eerst alles typeren ("dit is een kist met appels, dat zijn bananen") en dat is veel werk. Het vereist preciese kennis van de grammatica en dat werpt een drempel op. Het is dus een afweging tussen zekerheid, flexibiliteit, en productiviteit. Met een overwegend dynamisch getypeerde taal kun je weliswaar snel applicaties bouwen, maar het resultaat is gevoeliger voor fouten, dus je zult het eindproduct veel meer moeten testen. Het werk verschuift dan voor een groot deel van coderen naar testen.

### Gereedschap

Elke professional heeft behoefte aan goed gereedschap. Dat begint met een goede werkomgeving. Wat het wetboek en de jurisprudentie zijn voor de jurist, zijn bibliotheken en frameworks voor de programmeur. Voor het schrijven van code gebruikt de ontwikkelaar een geavanceerde editor die lijkt op Word. Hij vult automatisch tekst aan, controleert tijdens het intikken de syntax en compileert direct. Zo'n editor heet een Integrated Development Environment (IDE). Hij bevat helpfuncties, support en ingebouwde debuggers. Kortom een onmisbaar stuk gereedschap.

### Bibliotheek en framework

Een taal beschikt meestal ook over een standaard bibliotheek, waarin veelgebruikte functies al compleet zijn uitgewerkt. Deze functies kan de programmeur middels de API (Application Programmers Interface) gemakkelijk aan zijn werk toevoegen. Een standaard bibliotheek is een onderdeel van de taal. Daarnaast bestaan er frameworks. Dit zijn bibliotheken die ontwikkeld zijn voor een

speciaal doel. Ook weer met complete stukken functionaliteit die in één keer een bepaald concept neerzetten. Voor sommige talen zijn er enorme frameworks, zoals Java EE (Java Enterprise Edition).

De vaardigheid van een programmeur richt zich vooral op kennis hebben van die berg aan functionaliteit en weten waar je wat kunt vinden. Een goed framework, goede documentatie en goede ondersteuning vanuit een IDE werkt zeer productieverhogend.

### Populariteit en remmende voorsprong

De keuze voor een taal bepaalt welke frameworks en welke ontwikkelomgevingen je tot je beschikking hebt. Omgekeerd kunnen een goede IDE en een groot framework een taal populair maken. Maar de populariteit van een taal of een enorm framework kan ook afremmen. De taal wordt nauwelijks meer verbeterd of een framework heeft een zeer steile 'learning curve' en remt uiteindelijk de productiviteit toch weer af. Daar spelen nieuwe talen op in. Die kunnen van nul af aan beginnen en alle ballast overboord gooien. Vaak is een nieuwe taal min-of-meer een kopie van een bestaande succesvolle taal. Een mix van het goede uit andere talen, maar allemaal net iets handiger. Anderzijds verdwijnt daarmee ook ervaring en historie, inclusief een framework.

### Standaardiseren of vernieuwen?

De drijvende kracht achter de innovatie en evolutie van talen is het streven naar hogere productiviteit en kwaliteit. Soms kan dat met een goed en uitgebreid framework. Een andere keer gaat dat beter met een nieuwe taal. Een taal tot standaard bombarderen werkt niet, hooguit tijdelijk. Talen en frameworks verouderen. We noemen dit dan 'legacy'. Ze kunnen de race van alsmaar nieuwe hardware, software en besturingssystemen niet bijbenen. De onderhoudskosten van legacysystemen zullen steeds blijven oplopen. Het is een eeuwige vlucht voorwaarts waarin niemand kan blijven stilstaan. Bij spelcomputers accepteert men dat een spelletje van de Play Station 1 niet op de Play Station 3 draait. Maar een bedrijf dat voor zijn core business volledig van zijn IT-systemen afhankelijk is, zal toch af en toe gedwongen zijn om een ingrijpend besluit te nemen en een grote stap voorwaarts te maken.